Structures de Données

SMP S5

Automne 2011

M. M. Himmi - FSR

- Une structure de données est une organisation logique des informations destinée à simplifier leur traitement.
- Le choix d'utiliser une structure de données appropriée à un traitement informatique peut faire baisser de manière significative la complexité algorithmique.

2

Objectif du cours

- **Introduire quelques structures utilisées de façon intensive en programmation.**
- **Apprendre à gérer un ensemble fini d'éléments dont le nombre n'est pas fixé a priori**

3

Justification de la notion de type 1/2

- La mémoire centrale d'un ordinateur est un ensemble de **positions binaires** regroupées en **octets ou groupe d'octets**, repérée par une **adresse**.
- Toute information quelle que soit sa nature est **codée** sous forme **binaire**.
- Il ne suffit pas de connaître le contenu d'une zone pour lui attribuer une signification.

4

Justification de la notion de type 2/2

*Si un octet contient : **11001101** On peut considérer que cela représente le nombre entier **205** mais puisque tout est codé en binaire, cet octet peut aussi bien représenter un entier négatif, une partie du codage d'un réel ou autre…*

La notion de type sert (entre autres) à connaître le codage utilisé pour choisir les bonnes instructions.

5

Les types de base (Langage C)

La norme ANSI prévoit 6 types d'**entiers** qui agissent sur :

- La taille de l'emplacement mémoire pour représenter les valeurs:

  int ***short int*** ***long int***

- Le mode de représentation binaire:

  signed ***unsigned***

Ils permettent de représenter une partie des nombres entiers relatifs

6

Les types de base (Langage C)

Elle prévoit aussi 2 types de **caractères**:

signed char ***unsigned char***

Une variable du type char peut contenir une valeur entière entre -128 et 127 et elle peut subir les mêmes opérations que les variables du type *short*, *int* ou *long*

7

Les types de base (Langage C)

Avant d'utiliser une variable, nous devons nous intéresser à deux caractéristiques :

- le domaine des valeurs admissibles
- le nombre d'octets réservé pour une variable

Type	Description	Min	Max	Taille (octets)
char	caractère	-128	127	1
short	entier court	-32 768	32 767	2
int	entier standard	-32 768	32 767	2
long	entier long	-2 147 483 648	2 147 483 647	4

8

Les types de base (Langage C)

Si on ajoute le préfixe *unsigned* à la définition d'un type de variables entières, les domaines des variables sont déplacés comme suit:

Type	Description	Min	Max	Taille (octets)
unsigned char	caractère	0	255	1
unsigned short	entier court	0	65535	2
unsigned int	entier standard	0	65535	2
unsigned long	entier long	0	4 294 967 295	4

9

Les types de base (Langage C)

Pour les réels, la norme prévoit 3 types flottants:

float ***double*** ***long double***

Définition	Min	Max	Taille (octets)	Précision
float	$1.17 * 10^{-38}$	$3.4 * 10^{38}$	4	$1.19 * 10^{-7}$
double	$2.22 * 10^{-308}$	$1.79 * 10^{308}$	8	$2.22 * 10^{-16}$
long double	$3.36 * 10^{-4932}$	$1.18 * 10^{4932}$	10	$1.08 * 10^{-19}$

10

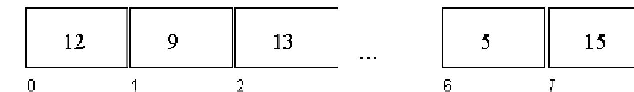
Types composés: Tableaux

- collection de *n* cellules mémoires consécutives, de même type (même taille) pas nécessairement simple.
- Les *n* cellules sont indexées, ce qui permet l'accès direct aux éléments du tableau en temps constant.

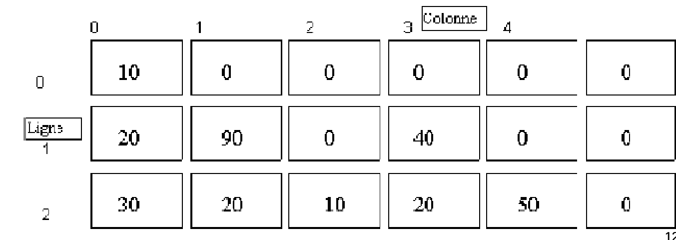
11

Types composés: Tableaux

- Un tableau à une dimension est un 'vecteur'



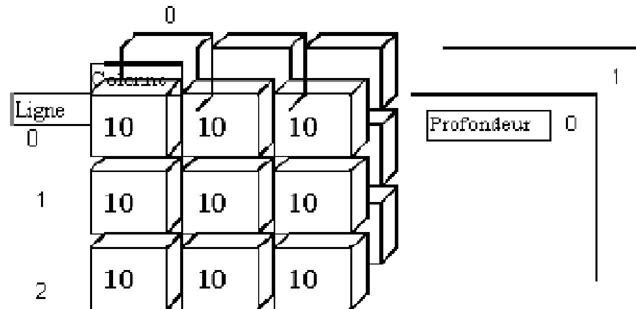
- Un tableau à deux dimensions est une 'matrice'



12

Types composés: Tableaux

- Tableau à trois dimensions



13

Types composés: Tableaux

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.
- Le tableau bidimensionnel (3 lignes, 4 colonnes) est en fait un tableau comportant 3 éléments, chacun d'entre eux étant un tableau de 4 éléments.

1	2	3	4
5	6	7	8
9	10	11	12

Il est stocké en mémoire de la manière suivante:

1	2	3	4	5	6	7	8	9	10	11	12

14

Types composés: Tableaux

Tableau à une dimension:

type nom_du_tableau [nombre d'éléments]

Exemple:

int t[10]; float notes[300];

Tableau à N dimensions:

type nom_du_tableau [a1][a2][a3] ... [an]

Exemple:

int t[10][2]; float notes[60][5];

15

Types composés: Structures

- Un tableau permet de regrouper des éléments de même type. Il est utile de pouvoir rassembler des éléments de type différent tels que des entiers, des chaînes de caractères, …
- Les structures permettent de regrouper des informations dans une entité repérée par un seul nom de variable.
- Les objets contenus dans la structure sont appelés champs de la structure.

16

Types composés: Structures

Exemple:

- Dans un répertoire téléphonique chaque fiche rassemble les informations d'une personne: Nom, Prénom, Adresse, date d'anniversaire, téléphone, Fax, email, …

17

Types composés: Structures

On peut définir ces objets par l'instruction:

```
struct fiche {
        char nom[20];
        char prenom[10];
        char adresse[40];
        char amj[8];
        long tel;
} personne;
```

18

Types composés: Structures

Ou si le type date est défini par la structure:

```
typedef struct date {
        int annee;   int mois;   int jour;
} date;

struct fiche {
        char nom[20]; char prenom[10];
        char adresse[40]; struct date ddn;
        char tel[10];
} personne;
```

19

Types composés: Structures

- Ainsi, on peut définir de nouveaux types de données avec des opérateurs spécialisés pour ces types, et donc étendre la puissance d'expression du langage

Questions:

- Quelles opérations peut-on définir avec le type '***date***'?
- La différence de deux dates est une durée, elle peut être représenté par un entier!
- Est il possible d'additionner deux dates?

20

Types composés: Structures

Exercice:

- ✓ écrire les fonctions
  - saisie_date(),
  - affiche_date(),
  - difference_date(),
  - calcule_date(date d, long duree)
- ✓ définir le type de donnée ‘heure’

21

Ensembles (Rappels)

Un ensemble est une collection d’objets, appelés éléments de l’ensemble.

Un ensemble est défini:

- Soit en intention: par une propriété vérifié par tous les éléments de l’ensemble:
  - Ensemble des nombres premiers
  - Ensemble des entiers impairs inférieurs à 9999
  - Ensemble des étudiants nés entre 1985 et 1990
- Soit en extension: énumération de tous les éléments de l’ensemble:
  - Alphabet = { a, b, c, d, e, …}
  - Jours de la semaine = { dimanche, lundi, …}

22

Opérations sur les ensembles

L’algèbre des ensembles repose sur quelques opérations élémentaires dont:

- L’appartenance d’un élément à un ensemble
- La réunion de deux ensembles
- L’intersection de deux ensembles
- Le complémentaire de deux ensemble
- Etc. …

➢ ***La notion d’ensemble n’implique pas nécessairement de relation d’ordre entre les éléments.***

23

Implémentation d’ensembles

➢ Les ensembles que l'on utilise en programmation sont des objets dynamiques; le nombre de leurs éléments varie au cours de l'exécution du programme puisqu'on peut ajouter et supprimer des éléments en cours du traitement.

24

Implémentation d'ensembles

➢ *Pour être efficace la gestion des ensembles doit répondre à deux critères parfois contradictoires:*

- *un minimum de place mémoire utilisée*
- *un minimum d'instructions élémentaires pour réaliser une opération..*

25

Structures séquentielles

- **Implémentation par Tableaux**

En reprenant le cas du répertoire téléphonique constitué de fiches ont peut définir:

struct fiche personne[200];

On est obligé de surdimensionné !

26

Structures séquentielles

- **Implémentation par Tableaux**

Considérer un tableau dynamique (allocation de mémoire *malloc(), calloc(), realloc()*)

```
struct  fiche *Tab;
Tab = malloc(200*sizeof(struct  fiche ));
```

Si 200 fiches ne suffisent plus on peut prendre un tableau plus grand. Mais il faut récupérer toutes les fiches du premier tableau …

27

Structures séquentielles

- **Implémentation par Tableaux**

<u>*Exemple*</u>

```
/* le type complexe */
   struct complexe {double Re; double Im; };
/* une définition plus courte pour la structure */
   typedef struct complexe Complexe;
   Complexe * tab;
   tab = malloc (3 * sizeof(Complexe));
   if( tab == NULL )
   {    printf("Allocation impossible");
        exit(); }
```

28

Structures séquentielles

Exemple (suite)
pour accéder aux trois cases du tableau:

```
/* La première case */
(*tab).Re = 10.;
(*tab).Im = 10.;
/* La deuxième case */
(*(tab + 1)).Re = 15.;
(*(tab+ 1)).Im = 20.;
/* La troisième */
(*(tab + 2)).Re = 1.;
(*(tab + 2)).Im = 1.;
```

29

Structures séquentielles

Vous trouvez cela compliqué et lourd à écrire ?
vous pouvez procédez comme suit :

```
/* La première case */
tab[0].Re = 10.;
tab[0].Im = 10.;
/* La deuxième case */
tab[1].Re = 15.;
tab[1].Im = 20.;
/* La troisième */
tab[2].Re = 1.;
tab[2].Im = 1.;
```

30

Structures séquentielles

- **Implémentation par Tableaux (statiques ou dynamiques !)**

- **a∈A** et **a∉A** : Vérifier l'appartenance d'un élément consiste à parcourir tout le tableau, soit ***n*** comparaisons s'il y a ***n*** fiches.

- Il faut avoir une fonction de comparaison adaptée aux données … !

31

Structures séquentielles

- **Implémentation par Tableaux**

- La réunion de deux ensembles **A∪B** peut se concevoir de différentes façons…, faire attention à la taille !

On parcourt les deux tableaux: N+M opérations

32

Structures séquentielles

- **Implémentation par Tableaux**

- L'intersection de deux ensemble **A∩B** nécessite de créer un nouvel ensemble !
  On parcourt le tableau A et on vérifie que l'élément ∈ B : NxM opérations
- Le complémentaire de A dans B (**B-A**): il faut avoir une fonction de suppression d'un élément d'un ensemble … ou créer un nouvel ensemble C des éléments de B n'appartenant pas à A

33

Structures séquentielles

- **Implémentation par Tableaux**

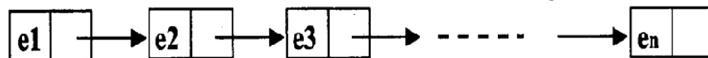
- La représentation par des tableaux est pratique pour les ensembles à taille réduite, déterminée à priori et n'évoluant pas beaucoup en cours d'exécution.
- Coût élevé de la recherche d'un élément pour les ensembles non ordonnés

34

Structures séquentielles

Listes chaînées

Une liste chaînée est une suite finie de cellules contenant chacune un élément de la liste et l'adresse de l'élément suivant dans la liste



Les cellules sont des objets dont un des champs contient une référence vers la cellule suivante.

35

Structures séquentielles

Listes chaînées

On définit le type cellule comme une structure dont le premier champ contient un élément de l'ensemble et le deuxième champ est un pointeur sur la cellule suivante

```
typedef struct cellule {
        int contenu;
        struct cellule *suivant ;
} liste;
```

36

Structures séquentielles

Listes chaînées

La fonction *alloc_cellule(x)* retourne un pointeur sur la cellule d'une liste dont la valeur est *x*.

```
liste * alloc_cellule (int x) {
      liste *c;
      c=(liste *) calloc (1, sizeof (liste));
      if (c!=NULL) {
            c->contenu=x;
            c->suivant=NULL;
      }
      return ( c );
}
```

37

Structures séquentielles

Listes chaînées: principales opérations

Liste vide?

```
int estvide (liste *l) {
  return (l==NULL);
}
```

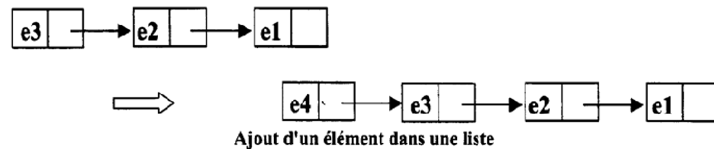
38

Structures séquentielles

Listes chaînées: principales opérations

Ajout d’un élément

Pour que le nombre d'opérations lors de l'ajout soit indépendant de la taille de la liste, la fonction *ajouter* insère l'élément *x* en tête de liste:



Ajout d'un élément dans une liste

39

Structures séquentielles

Listes chaînées: principales opérations

Ajout d’un élément

```
liste * ajouter (int x, liste *l) {
 liste *c;
 c=(liste *) calloc (1, sizeof (liste));
 if (c!=NULL){
        c->contenu=x;
        c->suivant=l;
        }
 return (c);
 }
```

40

Structures séquentielles

Listes chaînées: principales opérations

Recherche de l'élément *x*
On effectue un parcours de la liste:

```
int recherche (int x, liste *l) {
 while (l!=NULL) {
        if (l->contenu==x)
                return (1);
        l = l->suivant;
 }
 return (0);
 }
```

41

Structures séquentielles

Listes chaînées: principales opérations

Recherche de l'élément *x:*
La fonction recherche peut être récursive …

```
int recherche2 (int x, liste *l) {
 if (l== NULL)
        return (0);
else if (l->contenu == x)
                return (1);
         else
                return recherche2 (x, l->suivant);
}
```

42

Structures séquentielles

Listes chaînées: principales opérations

Longueur d'une liste: (récursive)

```
int longueur (liste *l) {
 if (l == NULL)
        return (0);
    else
        return (1 + longueur (l->suivant));
}
```

43

Structures séquentielles

Listes chaînées: principales opérations

Longueur d'une liste: (itérative)

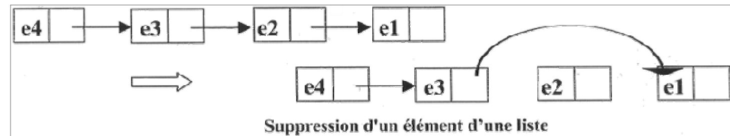
```
int longueur1 ( liste *l) {
 int longueur = 0;
 while (l != NULL) {
        ++longueur;
        l = l->suivant;
        }
 return longueur;
}
```

44

Structures séquentielles

Listes chaînées: principales opérations

La suppression de la cellule qui contient x s'effectue en modifiant la valeur de suivant contenue dans le prédécesseur de x:



Suppression d'un élément d'une liste

Le successeur de x devient le successeur du prédécesseur de x.

- **Un traitement particulier doit être fait si l'élément à supprimer est le premier élément de la liste!**

45

Structures séquentielles

Listes chaînées: principales opérations

Suppression: la fonction récursive est compacte:

```
liste* supprimer (int x, liste *l) {
    if (l != NULL)
        if (l->contenu == x)
            l = l->suivant;
        else
            l->suivant = supprimer (x, l->suivant);
    return ( l);
    }
```

- **On ne s'est pas préoccupé de la récupération de la mémoire !**

46

Structures séquentielles

Listes chaînées: principales opérations

Suppression d'une cellule:

- **La fonction itérative demande beaucoup plus d'attention essayez !**
- **Que se passe t-il si on supprime le seul élément d'une liste?**

47

Structures séquentielles

Listes chaînées:

- La procédure ajouter effectue 3 opérations élémentaires. Elle est donc très efficace. En revanche, les procédures recherche et supprimer sont plus longues puisqu'on peut aller jusqu'à parcourir la totalité d'une liste pour retrouver un élément.
- Si on ne fait aucune hypothèse sur la fréquence respective des recherches, que le nombre d'opérations est en moyenne égal à la moitié du nombre d'éléments de la liste.

48

Comparaisons Tableaux / Listes 1/2

Tableau:	Liste chaînée:
•la taille est connue	•la taille est inconnue au début
•l'adresse est connue.	•l'adresse est inconnue au début
•le stockage est contigu	•le stockage est éparpillé
•la taille des éléments est connue	•la taille des éléments est connue
•on peut atteindre directement la case i.	•impossible d'accéder directement à l'élément i.
•Pour déclarer un tableau: il faut connaître sa taille.	•Pour déclarer une liste chaînée créer le pointeur qui va pointer sur le premier élément.

49

Comparaisons Tableaux / Listes 2/2

Tableau:

•Pour supprimer ou ajouter un élément: on créer un nouveau tableau.

•L'adresse du premier élément du tableau peut changer, puisque realloc n'aura pas forcement la possibilité de trouver en mémoire la place nécessaire et contiguë pour allouer votre nouveau tableau.

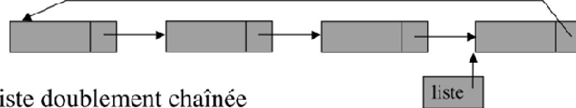
Liste chaînée:

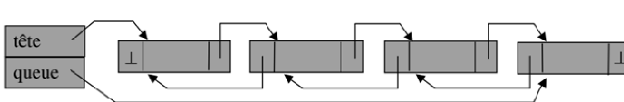
•On peut ajouter, supprimer, intervertir des éléments d'une liste chaînée sans avoir à recréer la liste. En manipulant simplement leurs pointeurs.

50

Structures séquentielles

Listes chaînées: on peut tout imaginer …





51

Structures séquentielles

Les Files *permettent de réaliser une FIFO (First In First Out): les premiers éléments ajoutés à la file sont les premiers à être récupérés.*

- *Elles sont utilisées en programmation pour gérer des objets qui sont en attente d'un traitement.*
- Les éléments sont ajoutés en queue et supprimés en tête
- La valeur d'une file est par convention celle de l'élément de tête

52

Structures séquentielles: Files

Les opérations sur les files satisfont les relations suivantes:

- **Pour $F \neq F_0$**

  supprimer(ajouter(x,F)) = ajouter(x ,supprimer(F))
  valeur(ajouter(x,F))= valeur(F)

- **Pour toute file F**

  estVide(ajouter(x,F)) = faux

- **Pour la file F_0**

  supprimer(ajouter (x,F_0))= F_0
  valeur(ajouter(x,F_0)) = x
  estVide(F_0) = vrai

53

Structures séquentielles

Exemple: File d'attente d'un guichet:
Chaque client qui se présente obtient un numéro et les clients sont ensuite appelés au guichet en fonction croissante de leur numéro d'arrivée.

- Deux nombres doivent être connus par les gestionnaires*:*
  - le numéro obtenu par le dernier client arrivé
  - le numéro du dernier client servi.
- On note ces deux nombres par *fin* et *début*

54

Structures séquentielles

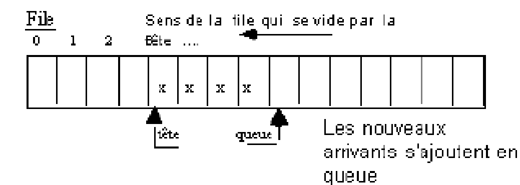
Exemple : *File d'attente d'un guichet*

- On gère le système de la façon suivante:
  - la file d'attente est vide si et seulement si *début = fin*
  - lorsqu'un nouveau client arrive on incrémente *fin* et on donne ce numéro au client.
  - lorsque le serveur est libre et peut servir un autre client, si la file n'est pas vide, il incrémente *début* et appelle le possesseur de ce numéro.

55

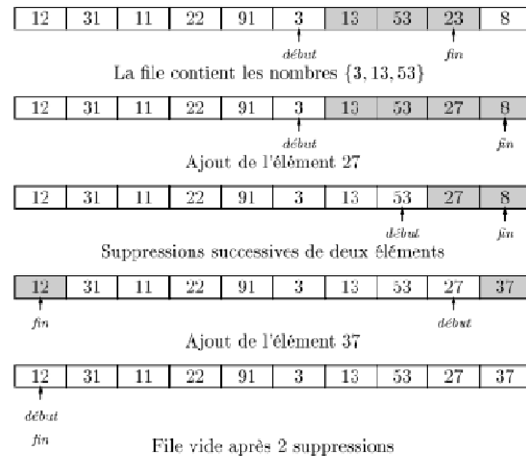
Structures séquentielles

Exemple : *File d'attente d'un guichet*
Implémentation par tableau: On dispose de deux index et d'un tableau de taille MAXELEM



56

Structures séquentielles

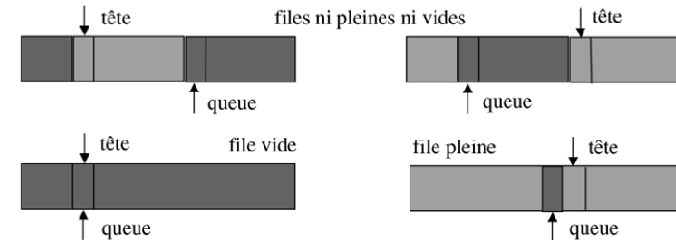


57

Structures séquentielles

- **ambiguïté plein/vide:**
  toujours un emplacement vide au moins correspondant à l'élément servie

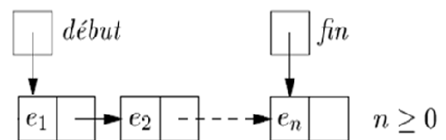
exemples de configurations:



58

Structures séquentielles

- Une autre façon de gérer des files consiste à utiliser des listes chaînées dans lesquelles on connaît à la fois l'adresse du premier et du dernier élément:



File d'attente implémentée par une liste

59

Structures séquentielles

- Opérations
  - Estvide
  - Premier
  - Ajouter
  - Retirer

60

Structures séquentielles

Les Piles

- La notion de pile intervient couramment en programmation son rôle principal consiste à implémenter les appels de procédures.
- On peut imaginer une pile comme une boîte dans laquelle les objets sont les uns sur les autres dans la boîte et on ne peut accéder qu'à l'objet situé au "sommet de la pile"

61

Structures séquentielles

Les Piles

Les relations satisfaites par les piles sont:

Si (P_0 est la pile vide)

- supprimer (ajouter (x,P)) = P
- estVide (ajouter (x,P)) = faux
- valeur (ajouter (x,P)) = x
- estVide (P_0) = vrai

62

Structures séquentielles

Les Piles

- On peut exprimer toute expression sur les piles faisant intervenir ces 4 opérations à l'aide de la seule opération ajouter en partant de la pile P_0 .

Ainsi:

Supprimer (ajouter (7 ,supprimer (ajouter (valeur (ajouter (5, ajouter (3, P_0)))),ajouter (9, P_0))))

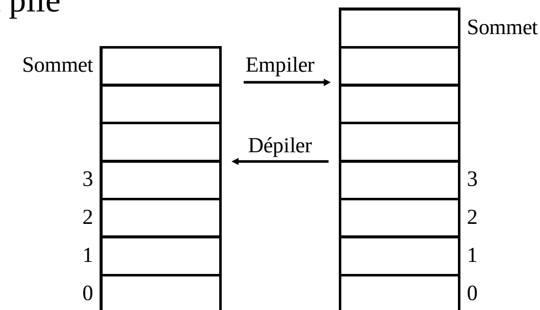
peut se simplifier en:

ajouter (9, P_0)

63

Structures séquentielles

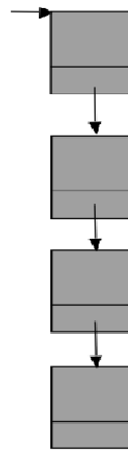
- Les opérations sur les piles peuvent s'effectuer en utilisant un tableau qui contient les éléments et un indice qui indiquera la position du sommet de la pile



64

Structures séquentielles

- ou avec une liste chaînée:



65

Structures séquentielles

- Opérations
  - Estvide
  - Sommet
  - Empiler
  - Dépiler

66

- Soit un serveur d'impression où on affecte à chaque requête d'impression un nombre, plus ce numéro est élevé plus le document doit être imprimé rapidement
- Le serveur doit donc faire rapidement les trois opérations suivantes:
  - trouver un maximum
  - retirer cet élément
  - ajouter un nouvel élément

Question: Comment implémenter une telle tâche?

68

- Prendre un tableau, le trier dans l'ordre croissant des priorités. Trouver un maximum et le retirer est alors facile. L'insertion est plus compliquée …
- Gérer une file simple, rechercher le maximum à chaque fois, le retirer …

ou

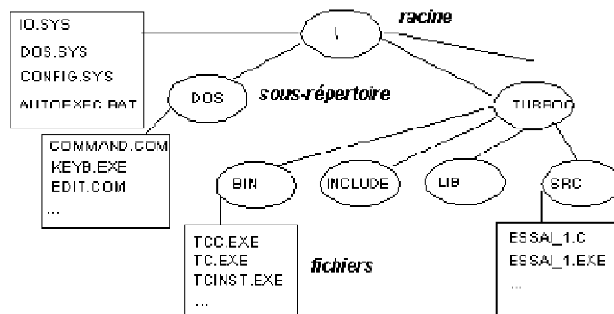
- Considérer une structure à ordre partiel:

un arbre

69

Structures arborescentes

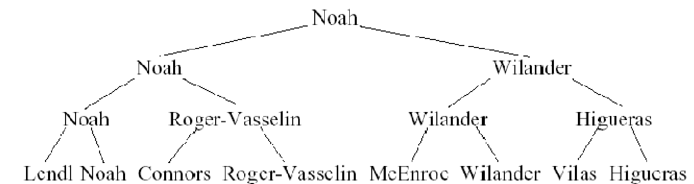
Collections hiérarchisées: Exemples



70

Structures arborescentes

Collections hiérarchisées



71

Structures arborescentes

Quelques définitions:

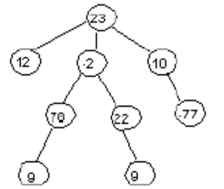
Arbre vide

Arbre singleton (23)

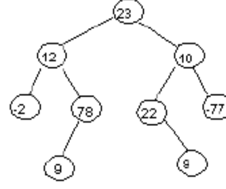
Arbre filiforme



Arbre quelconque



Arbre binaire

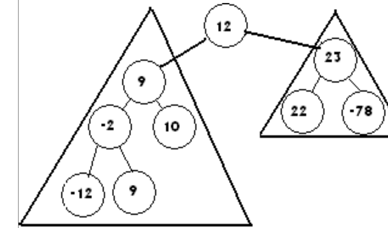


72

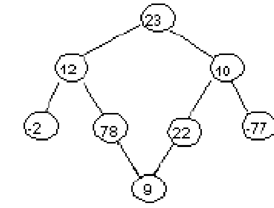
Structures arborescentes

Quelques définitions:

Arbre binaire de recherche



CECI N'EST PAS UN ARBRE!

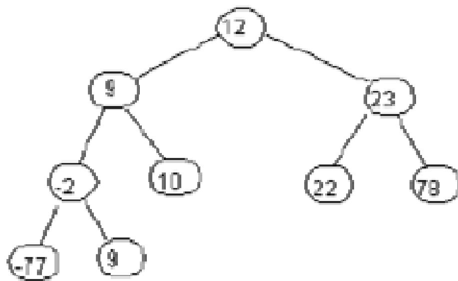


73

Structures arborescentes

Quelques définitions:

Arbre binaire de recherche



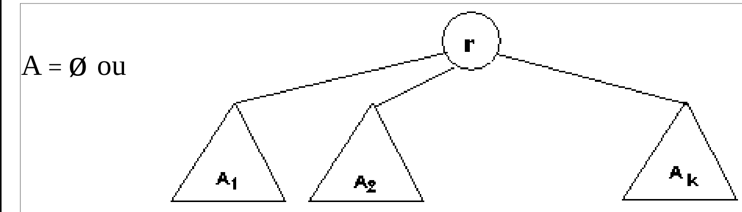
74

Structures arborescentes

Quelques définitions:

- C'est une structure récursive: un arbre est constitué d'un nœud appelé racine et de sous arbres fils de la racine.

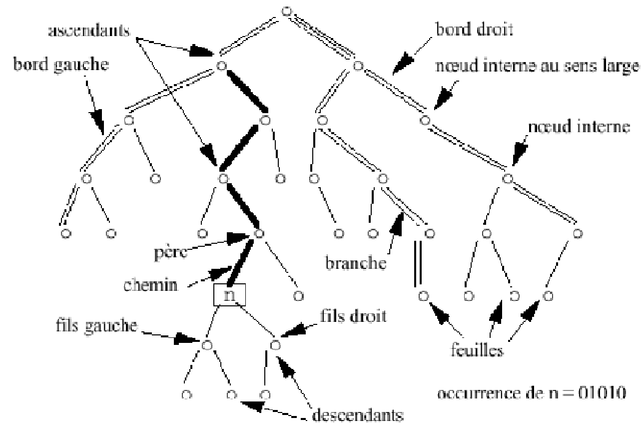
A = Ø ou



75

Structures arborescentes

Terminologie



76

Arbres: Caractéristiques

- Nombre de nœuds:

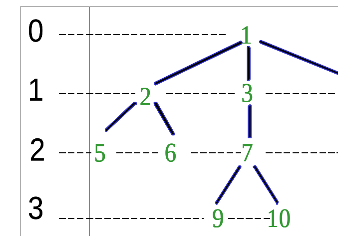
  Taille(A) = 0 si A=ø

  1+taille(A1)+taille(A2) si A=<o,A1,A2>

- Nombre de feuilles d'un arbre:

  Nf(A) = 0 si A=ø

  max(1,Nf(A1)+Nf(A2)) sinon



77

Arbres: Caractéristiques

- Hauteur d'un nœud: distance de x à la racine

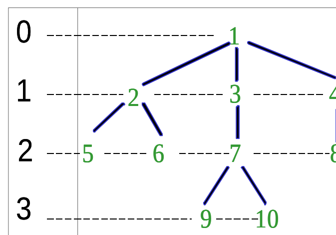
  h(x) = 0 si x = racine(A)

  1 + niveau (parent(x)) sinon

- Hauteur d'un arbre:

  H(A) = -1 si A=ø

  1+max(h(A1),h(A2)) sinon



78

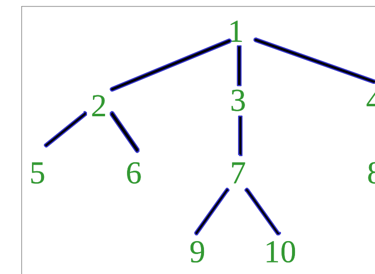
Arbres : Parcours en profondeur

$A = (r, A_1, A_2, \ldots, A_k)$

- Parcours préfixe

  $P(A) = (r).P(A_1). \ldots .P(A_k)$

  (1, 2, 5, 6, 3, 7, 9, 10, 4, 8)



79

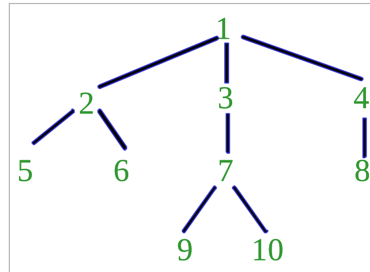
Arbres : Parcours en profondeur

$A = (r, A_1, A_2, \ldots, A_k)$

- Parcours suffixe

  $S(A) = S(A_1). \ldots .S(A_k).(r)$

  (5, 6, 2, 9, 10, 7, 3, 8, 4, 1)



80

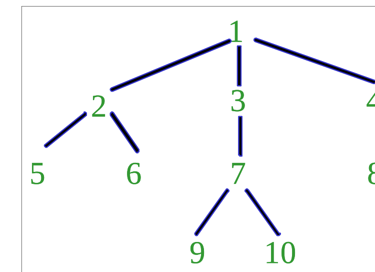
Arbres : Parcours en profondeur

$A = (r, A_1, A_2, \ldots, A_k)$

- Parcours symétrique (ou interne ou infixe)

  $I(A) = I(A_1).(r).I(A_2). \ldots .I(A_k)$

  (5, 2, 6, 1, 9, 7, 10, 3, 8, 4)



81

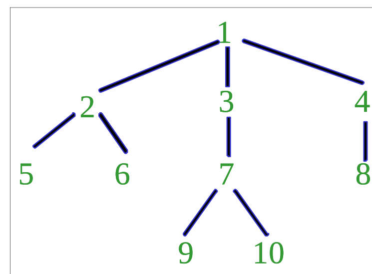
Arbres : Parcours en largeur

$A = (r, A_1, A_2, \ldots, A_k)$

H(A) = (r, x1, …, xi, xi+1, ... xj, xj+1, ..., xn)

nœuds de niveau 0, 1, 2, …

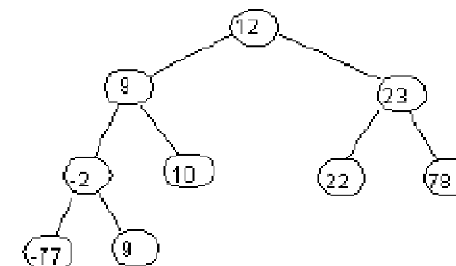
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)



82

Arbres : Parcours en profondeur

Cas d'un arbre binaire de recherche:



Parcours préfixe : 12, 9, -2, -77, 9, 10, 23, 22, 78

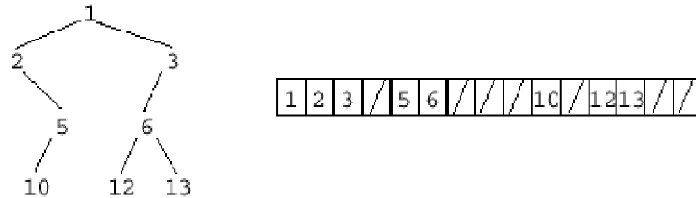
Parcours infixe : -77, -2, 9, 9, 10, 12, 22, 23, 78

Parcours suffixe : -77,9, -2, 10, 9, 22, 78, 23, 12

83

Arbres: Implémentations

Peut se faire avec des tableaux: …



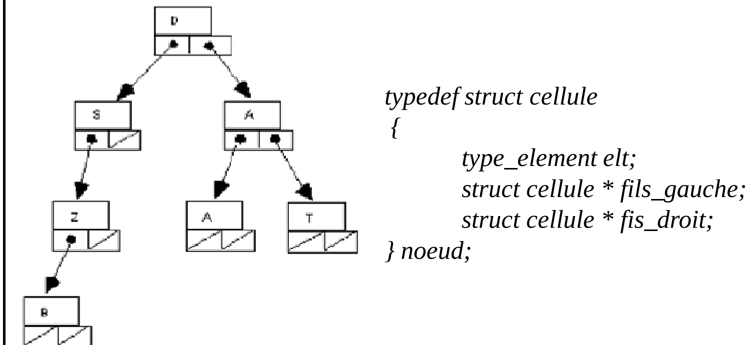
… Pas conseillé !

84

Arbres: Implémentations

Il vaut mieux voir du coté de l'allocation dynamique:

- Pour les arbres binaires pas de problèmes …

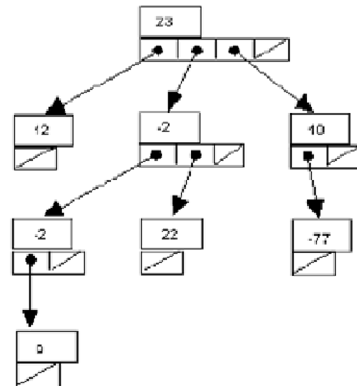


typedef struct cellule
{
type_element elt;
*struct cellule * fils_gauche;*
*struct cellule * fis_droit;*
} noeud;

85

Arbres: Implémentations

- Pour les arbres généraux, c'est plus délicat!

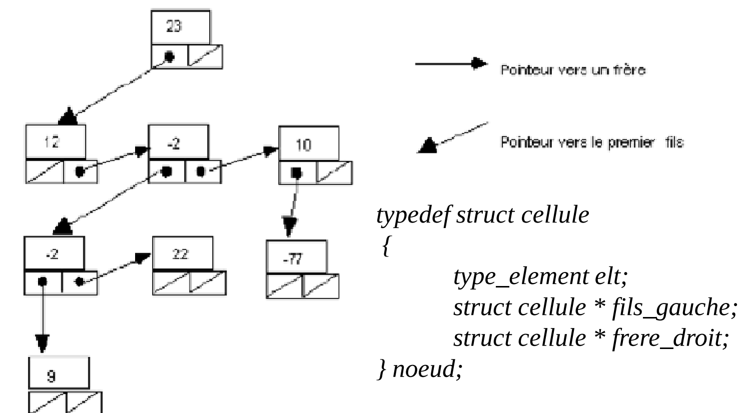


Problème:
Nombre variable de pointeurs par nœud …

86

Arbres: Implémentations

- Mais on peut envisager la structure:

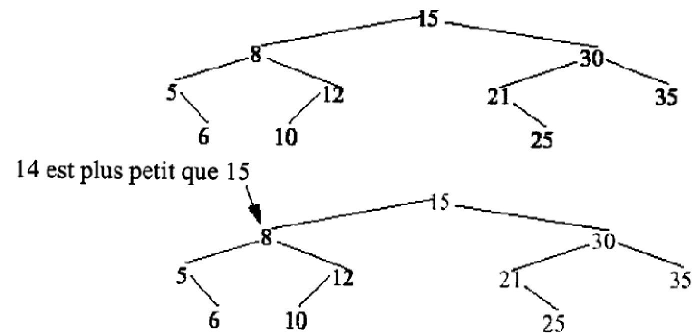


typedef struct cellule
{
type_element elt;
*struct cellule * fils_gauche;*
*struct cellule * frere_droit;*
} noeud;

87

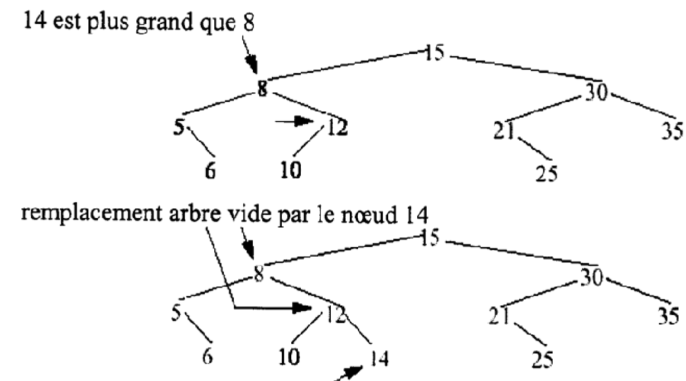
Structures arborescentes

Insertion d'une feuille:
Soit à ajouter "14" dans l'arbre binaire de recherche suivant



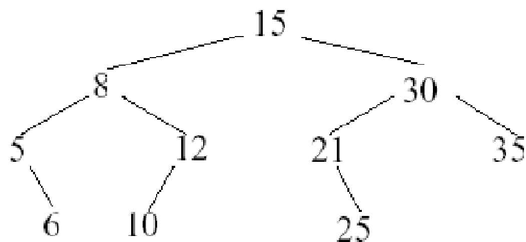
Structures arborescentes

Insertion d'une feuille: suite
Soit à ajouter "14" dans l'arbre binaire de recherche suivant



Structures arborescentes

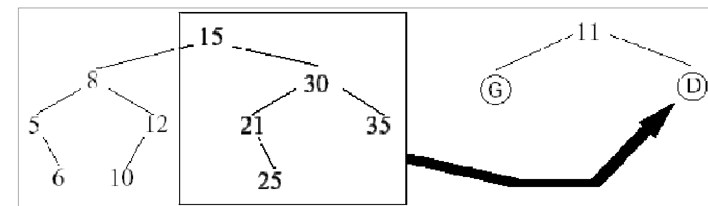
adjonction à la racine
Soit à ajouter "11" à l'arbre binaire de recherche suivant en tant que racine.



Structures arborescentes

adjonction à la racine

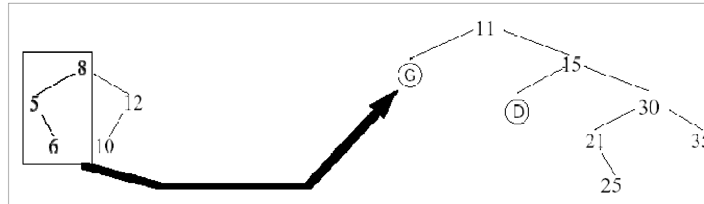
- 15 est plus grand que 11 donc à droite de 11
- ainsi que tous les neuds du sous-arbre droit de 15
- ceux qui restent sont plus petits que 15



Structures arborescentes

adjonction à la racine

- 8 est plus petit que 11
- Ainsi que tous ceux du sous-arbre gauche de 8
- ceux qui restent sont plus petits que 15 et plus grands que 8

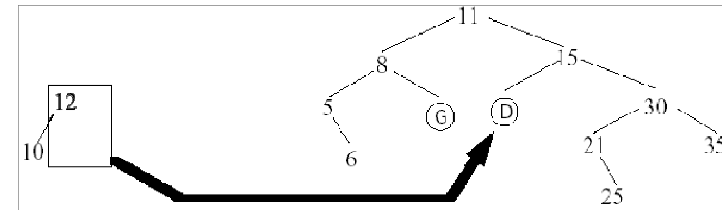


92

Structures arborescentes

adjonction à la racine

- 12 est plus grand que 11 (et plus petit que 15)
- Ainsi que ceux de son sous-arbre droit
- ceux qui restent sont plus grands que 8 et plus petits que 12

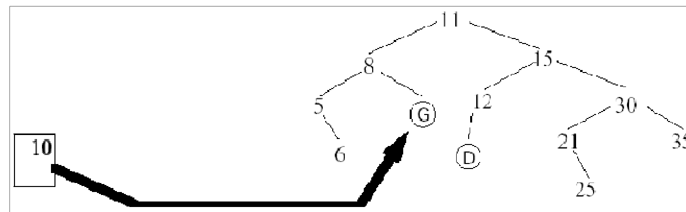


93

Structures arborescentes

adjonction à la racine

- 10 est plus petit que 11 (et plus grand que 8)
- Ainsi ceux de son sous-arbre gauche
- ceux qui restent sont plus grands que 10 et plus petits que 12

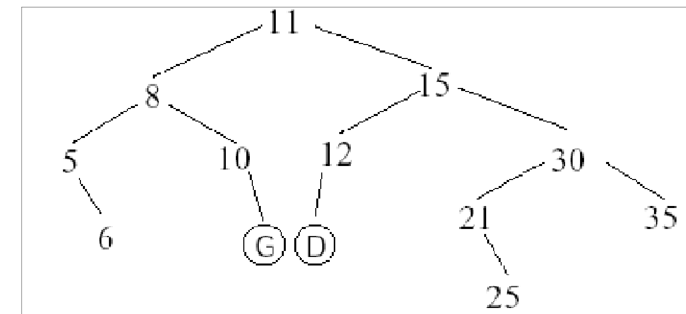


94

Structures arborescentes

adjonction à la racine

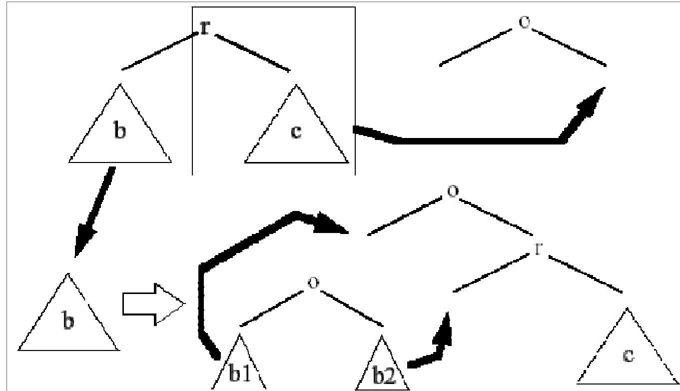
- l'arbre restant est vide: les arbres G et D sont vides.



95

Structures arborescentes

adjonction à la racine: Généralisation (cas o < r)



ajout(o, <r, b, c>) = <o, g(ajout(o, b), <r, d(ajout(o, b)), c>>

96

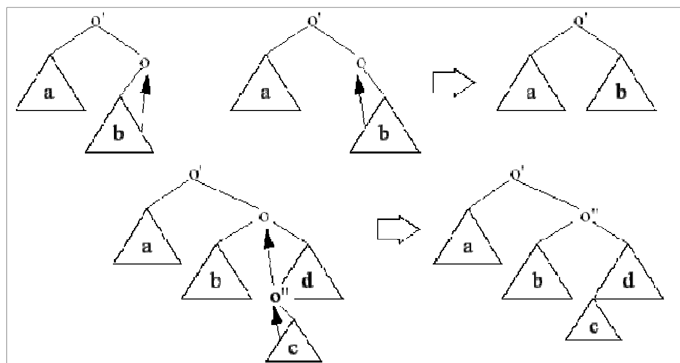
Structures arborescentes

- adjonction en feuille:
  les nœuds les plus anciens sont proches de la racine, et sont retrouvés plus vite.
- adjonction en racine:
  les nœuds les plus récents sont proches de la racine, et sont retrouvés plus vite.

97

Structures arborescentes

Suppression
rechercher le nœud, puis le supprimer: 3 configurations:



98

Structures arborescentes

- La complexité des ajouts et des suppressions dépends de la hauteur de l'arbre.
- Les ajouts et les suppressions aléatoires de nœuds peuvent aboutir à des hauteurs des nœuds très différentes entres les branches.
- Si on refuse les cas défavorables on peut rééquilibrer les arbres …

99

Structures arborescentes

Arbres H-équilibrés

A étant un arbre binaire; on définit:

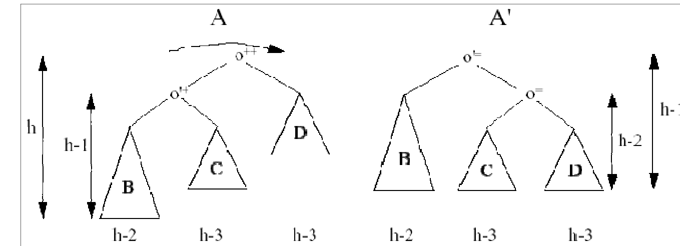
Déséquilibre(A)=	0	si A=∅
	h(g(A))-h(d(A))	sinon

A est un arbre H-équilibré si:
pour tout b sous arbre de A
déséquilibre(b) $\in$ {-1,0,1}

100

Structures arborescentes

Exemple de rééquilibrage: Rotation à droite



A arbre binaire de recherche => A' arbre binaire de recherche

Si B, C, D sont H-équilibré => A' est H-équilibré

101

Comment représenter et étudier:

- réseau de transport
- réseau informatique
- réseau électrique
- ordonnancement de tâches
- application hyper-media
- jeux
- ...

101

Structures arborescentes

Définition et vocabulaire

Un graphe G est un couple (S,A) où:

- *S est un ensemble FINI de sommets*
- *A est un sous ensemble de SxS; ensemble des arcs de G*

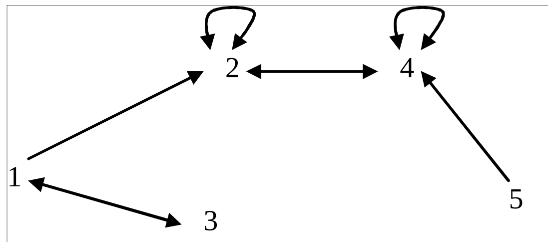
- ***Pour simplifier on prendra pour S l'intervalle [1,n].***

102

Structures arborescentes

- graphe orienté

G=(S1,A1)= ({1,2,3,4,5},{(1,2),(1,3),(2,2),(2,4),
(3,1),(4,2),(4,4),(5,4)})



a=(x,y) représente une liaison orienté entre l'origine x et l'extrémité y

or(a)=x   ***ext(a)=y***

103

Structures arborescentes

Si *a=(x,y)* on dit aussi que

- x est le ***prédécesseur*** de y
- y est le ***successeur*** de x
- x et y sont ***adjacents***
- si x=y l'arc est un ***boucle***

- Un **chemin *f*** du graphe ***G=(S,A)*** est une suite finie d'arcs $a_1, a_2, \ldots, a_p$ telle que:

$$\forall\ i\ ,\ 1< i <p\ or(a_{i+1}) = ext(a_i)$$

P est alors la longueur du chemin

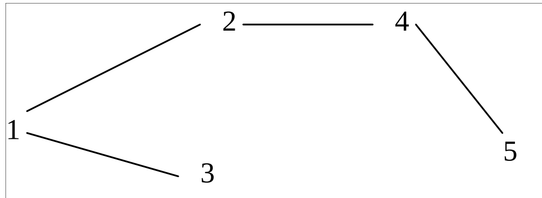
- Si or(f) = ext(f) le chemin est un **circuit**

104

Structures arborescentes

- G=(S,A) est ***non orienté si et seulement si***
  $\forall$ x,y Є S; (x,y) Є A ➔ (y,x) Є A

La représentation de l'exemple précédent devient:



105

Structures arborescentes

Dans le cas de graphes non orientés:

Les arcs s'appellent alors des ***arêtes*** et sont représentées par des paires {x,y}

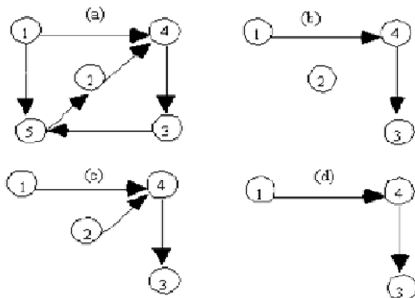
Le chemin devient une **chaîne**

et un circuit un **cycle**

106

Structures arborescentes

- G'=(S',A') est un sous graphe de G=(S,A):
  si S' $\subset$ S et A' $\subset$ A



(a) Graphe orienté G (b) Un sous-graphe de G
(c) Le graphe partiel induit par les sommets {1, 2, 3, 4}
(d) Le sous-graphe induit par les arcs {(1,4), (4,3)}

107

Structures arborescentes

Implémentation d'un Graphe

Trois représentations majeures:

- **la matrice d'adjacence**
- **la matrice d'incidence**
- **la liste des successeurs**

108

Implémentation d'un Graphe

1 - la matrice d'adjacence

- On associe à un graphe G=([1,n],A) une matrice carré d'ordre n dont les coefficients $M_{i,j}$ sont :

$M_{i,j}$ =1 si (x_i , x_j) € A et $M_{i,j}$ =0 sinon

Exemples:

G1 = ({1,2,3,4,5}, {(1,2), (1,3), (2,2), (2,4), (3,1), (4,2), (4,4), (5,4)})

M_1	1	2	3	4	5
1	0	1	1	0	0
2	0	1	0	1	0
3	1	0	0	0	0
4	0	1	0	1	0
5	0	0	0	1	0

Implémentation d'un Graphe

1 - la matrice d'adjacence

G2 = ({1,2,3,4,5}, {(1,5), (2,3), (2,4), (3,2), (3,5), (4,2), (4,5), (5,1), (5,3), (5,4)})

M_2	1	2	3	4	5
1	0	0	0	0	1
2	0	0	1	1	0
3	0	1	0	0	1
4	0	1	0	0	1
5	1	0	1	1	0

Si graphe non orienté la matrice d'adjacence est symétrique

Implémentation d'un Graphe

1 - la matrice d'adjacence

Théorème:

*Soit M^p la puissance p de la matrice **M**, le coefficient $M^p_{i,j}$ est égal au nombre de chemins de longueur **p** de **G** dont l'origine est le sommet x_i et l'extrémité est x_j.*

La Vérification de l'existence de chemins entre deux sommets revient au calcul des puissances successives de la matrice d'adjacence (forcément < nombre de sommets)

Implémentation d'un Graphe

2 - la matrice d'incidence

Si G est un graphe sans boucles (S,A), I(G) est une matrice SxA dans {-1,0,1} telle que l'élément $d_{x,a}$ =

1 si x est l'origine de l'arc a,

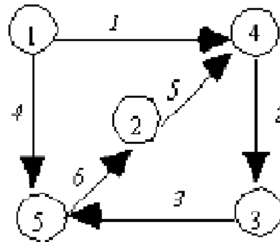
-1 si x est l'extrémité de l'arc a

0 sinon

Implémentation d'un Graphe

2 - la matrice d'incidence

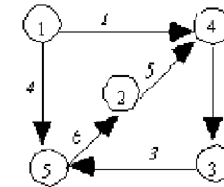
Exemple : matrices d'adjacence et d'incidence de G3.
G3 = (S3, A3) avec S3= {**1,2,3,4,5**} liste des sommets
et A3= {*1,2,3,4,5,6* } liste des arcs numérotés *1*=(1,4); *2*=(4,3) ; *3*=(3,5) ; *4*=(1,5) ; *5*=(2,4) ; *6*=(**5**,2)



113

Implémentation d'un Graphe

2 - la matrice d'incidence



M_3	1	2	3	4	5
1	0	0	0	1	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	1	0	0
5	0	1	0	0	0

I_1	1	2	3	4	5	6
1	1	0	0	1	0	0
2	0	0	0	0	1	-1
3	0	-1	1	0	0	0
4	-1	1	0	0	-1	0
5	0	0	-1	-1	0	1

114

Implémentation d'un Graphe

3 - liste des successeurs:

à chaque sommet x on associe la liste de ses successeurs

- Cette représentation est utile pour obtenir tous les successeurs d'un sommet x.

Si dans un graphe de 20000 sommets où chaque sommet n'a que 5 successeurs l'obtention de tous les successeurs de x se fait en consultant 4 ou 5 valeurs au lieu des 20000 tests à effectuer dans le cas des matrices.

115

Implémentation d'un Graphe

3 - liste des successeurs

- à l'aide d'un tableau à double indice:

Succ[x,i] est le i ème successeur de ***x****.*

- L'utilisation d'un symbole supplémentaire noté ω n'appartenant pas à X, permet une gestion plus facile

- Ainsi

Succ[x,i] = y Є X signifie que y est le $i^{ème}$ successeur de x

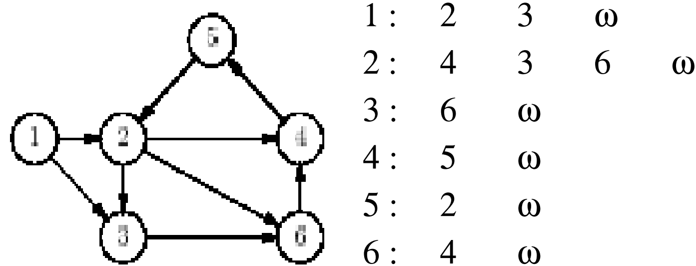
Succ[x,i] = ω signifie que x a (i-1) successeurs.

116

Implémentation d'un Graphe

3 - liste des successeurs

- à l'aide d'un tableau à double indice:



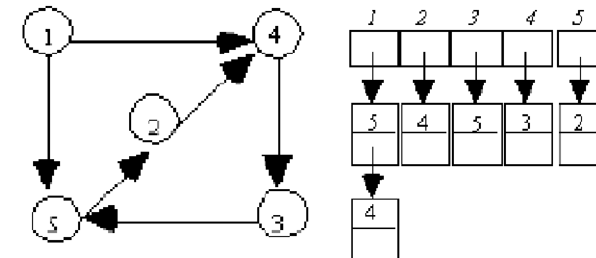
1 :	2	3	ω	
2 :	4	3	6	ω
3 :	6	ω		
4 :	5	ω		
5 :	2	ω		
6 :	4	ω		

117

Implémentation d'un Graphe

3 - liste des successeurs

- à l'aide d'un tableau de pointeurs sur la liste des successeurs de chaque sommet:



118

Le Hachage

Prenons le répertoire téléphonique personnel:

- Un certain nombre de pages réservées pour chaque lettre de l'alphabet.
- Les insertions se font au hasard … en fonction de la première lettre du nom (par exemple…)
- ➔ Gain de temps lors de la recherche…

Généralisation:
Comment faire des opérations de recherches et d'insertions dans une table en temps constant ?

119

Le Hachage

Réponse:

- établir une relation entre l'élément et l'adresse à laquelle il est rangé en mémoire …

- Si on devait insérer dans une table les éléments de l'ensemble [0,N-1]
  - il suffirait d'utiliser un tableau T de taille N
  - chaque élément peut alors être rangé dans la cellule correspondante h(m)=i

120

Le Hachage

Question:

- si on doit traiter un ensemble E de m entiers, comment construire une fonction injective de E dans [0,N-1] ? (*surtout si m > N !*)

Réponse:

Mathématiquement impossible! Mais on s'arrange pour que la fonction répartisse les clés de façon *"uniforme"*.

121

Le Hachage

- Deux critères principaux de choix d'une fonction de hachage H:
  - être facile et rapide à calculer
  - autant que possible distribuer les adresses calculées sur tout l'ensemble (minimum de collision)

122

Le Hachage

- Idées générales pour fabriquer des fonctions de hachage:
- Extraction de bits:
  extraire p bits de la clé, avec $m = 2^p$
- Compression de bits:
  découpage de la clé en q tranches de p bits puis combiner les tranches par xor et rotations
- Division:
  h(c) = c % m
- Autres …

Donc pas de fonction universelle il faut essayer sur échantillons !

123

Le Hachage

Exemples:

Prenons un ensemble de 100 adresses de 2 chiffres: 00, 01, …, 99 et appliquons quelques fonctions de hachage à chacun des objets à gérer suivants:

3205 7148 2345

- Méthode de division:

H(k)=k%m ou H(k)=k%m+1

Prenons un nombre m premier proche de 99 soit 97 alors:

H(3205)=4 H(7148)=67 H(2345)=17

124

Le Hachage

Méthode du mi-carré: H(k)=b où b est obtenu en supprimant des chiffres aux deux extrémités de k²

k:	3205	7148	2345
k²:	10272025	53093904	5499025
H(k):	72	93	99

Méthode de pliage H(k)=k1+k2+…+kr

La clé k est partitionnée en plusieurs parties k1,k2,…,kr de la même longueur que le nombre de chiffre de l'adresse

H(3205) = 32 + 05 = 37 H(7148) = 71 + 48 = 19

H(2345) = 23 + 45 = 68

Le Hachage

:

H(2345) = 23 + 45 = 68 et H(3632) = 36 + 32 = 68

on dit qu'il y a collision

Problème:

- Que faire dans ce cas?
- Où et comment stoker l'information et comment la relier à sa clé?

Le Hachage

2 Familles de solutions:

- A une autre place libre du tableau T
- Par calcul d'une autre fonction de hachage
- Première case vide suivante circulairement
- Autres

C'est un algorithme qui détermine la suite …

- Dans une zone à part: Zone de débordement
- Par pointeurs (listes chainée, arbres,…)
- Autres

C'est le hachage ouvert …

Le Hachage

Opérations (cas du hachage ouvert)

L'opération Recherche s'effectue en trois étapes:

1. Calcul de H(k): cela nécessite un temps constant.
2. Accès a la tête de la liste t[i]: en temps également constant
3. Recherche dans la liste: c'est le temps de cette dernière étape qu'il faut cerner.

Le Hachage

Si **n** est le nombre d'enregistrements et **m** est la longueur du tableau support, Dans le pire cas, la liste dont la tête est en t[i] pourrait être de longueur **n**.

- Une fonction de hachage correcte doit nous épargner en pratique cette calamitée.

129

Le Hachage

Si n est le nombre d'enregistrements et m est la longueur du tableau support:

On introduit la quantité **a = n/m** appelée facteur de remplissage, il peut être inférieur, égal ou supérieur a 1.

La recherche dans une table de hachage requiert en moyenne et en pratique un temps constant plus un temps proportionnel au facteur de remplissage.

130

Le Hachage

Pour un bon fonctionnement de la table de hachage, on s'arrange; pour que le facteur de remplissage reste constant et proche de 1, ce qui assure un temps de recherche moyen constant.

Si le facteur de remplissage augmente trop, suite a de nombreuses insertions, on doit reconstruire la table de hachage avec un nouveau tableau support de longueur plus grande, ce qui diminue le facteur de remplissage.

131

Le Hachage

Opérations

- Adjonction, Suppression: ➔ Rien de spécial

Autres utilisations du hachage:

- Signature de documents électroniques
- Cryptographie

132